Scheduling with Explorable Uncertainty*†

Christoph Dürr[1], Thomas Erlebach‡[2], Nicole Megow[3], and Julie Meißner[4]

1 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, Paris, France**
christoph.durr@lip6.fr
2 **Department of Informatics, University of Leicester, Leicester, UK**
te17@leicester.ac.uk
3 **Department of Mathematics and Computer Science, University of Bremen, Bremen, Germany**
nicole.megow@uni-bremen.de
4 **Institute of Mathematics, Technical University of Berlin, Berlin, Germany**
jmeiss@math.tu-berlin.de

Abstract

We introduce a novel model for scheduling with explorable uncertainty. In this model, the processing time of a job can potentially be reduced (by an *a priori* unknown amount) by testing the job. Testing a job j takes one unit of time and may reduce its processing time from the given upper limit $\bar{p}_j$ (which is the time taken to execute the job if it is not tested) to any value between 0 and $\bar{p}_j$. This setting is motivated e.g. by applications where a code optimizer can be run on a job before executing it. We consider the objective of minimizing the sum of completion times on a single machine. All jobs are available from the start, but the reduction in their processing times as a result of testing is unknown, making this an online problem that is amenable to competitive analysis. The need to balance the time spent on tests and the time spent on job executions adds a novel flavor to the problem. We give the first and nearly tight lower and upper bounds on the competitive ratio for deterministic and randomized algorithms. We also show that minimizing the makespan is a considerably easier problem for which we give optimal deterministic and randomized online algorithms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems – Sequencing and scheduling, F.1.2 Modes of Computation – Online computation, G.3 Probability and Statistics – Probabilistic algorithms (including Monte Carlo)

Keywords and phrases online scheduling, explorable uncertainty, competitive ratio, makespan, sum of completion times

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.30

1 Introduction

Uncertainty in scheduling has been modeled and investigated in many different ways, particularly in the frameworks of online optimization, stochastic optimization, and robust optimization. All these different approaches have the common assumption that the uncertain information, e.g., the processing time of a job, cannot be explored before making scheduling

* This research was carried out in the framework of MATHEON supported by Einstein Foundation Berlin, the German Science Foundation (DFG) under contract ME 3825/1 and the Bayerisch-Französisches Hochschulzentrum (BFHZ).
† A full version of this paper is available at [2], https://arxiv.org/abs/1709.02592
‡ The second author was supported by a study leave granted by University of Leicester.

decisions. However, in many applications there is the opportunity to gain exact or more precise information at a certain additional cost, e.g., by investing time, money, or bandwidth.

In this paper, we introduce a novel model for scheduling with explorable uncertainty. Given a set of n jobs, every job j can optionally be *tested* prior to its execution. A job that is executed without testing has processing time $\bar{p}_j \in \mathbb{Q}^+$, while a tested job has processing time p_j with $0 \leq p_j \leq \bar{p}_j$. Testing a job takes one unit of time on the same resource (machine) that processes jobs. Initially the algorithm knows for each job j only the upper limit $\bar{p}_j$, and gets to know the time p_j only after a test. Tested jobs can be executed at any time after their test. An algorithm must carefully balance testing and execution of jobs by evaluating the benefit and cost for testing.

We focus on scheduling on a single machine. Unless otherwise noted, we consider the sum of completion times as the minimization objective. We use competitive analysis to assess the performance of algorithms.

For the standard version of this single-machine scheduling problem, i.e., without testing, it is well known that the Shortest Processing Time (SPT) rule is optimal for minimizing the sum of completion times. The addition of testing, combined with the fact that the processing times p_j are initially unknown to the algorithm, turns the problem into an online problem with a novel flavor. An algorithm must decide which jobs to execute untested and which jobs to test. Once a job has been tested, the algorithm must decide whether to execute it immediately or to defer its execution while testing or executing other jobs. At any point in the schedule, it may be difficult to choose between testing a job (which might reveal that it has a very short processing time and hence is ideally suited for immediate execution) and executing an untested or previously tested job. Testing a job yields information that may be useful for the scheduler, but may delay the completion times of many jobs. Finding the right balance between tests and executions poses an interesting challenge.

If the processing times p_j that jobs have after testing are known, an optimal schedule is easy to determine: Testing and executing job j takes time $1+p_j$, so it is beneficial to test the job only if $1+p_j < \bar{p}_j$. In the optimal schedule, jobs are therefore ordered by non-decreasing $\min\{1+p_j, \bar{p}_j\}$. In this order, the jobs with $1+p_j < \bar{p}_j$ are tested and executed while jobs with $1+p_j \geq \bar{p}_j$ are executed untested. (For jobs with $1+p_j = \bar{p}_j$ it does not matter whether the job is tested and executed, or executed untested.)

Motivation and applications. Scheduling with testing is motivated by a range of application settings where an operation that corresponds to a test can be applied to jobs before they are executed. We discuss some examples of such settings. First, consider the execution of computer programs on a processor. A test could correspond to a code optimizer that takes unit time to process the program and potentially reduces its running-time. The upper limit of a job describes the running-time of the program if the code optimizer is not executed.

As another application, consider the transmission of files over a network link. It is possible to run a compression algorithm that can reduce the size of a file by an *a priori* unknown amount. If a file is incompressible (e.g., if it is already compressed), its size cannot be reduced at all. Running the compression algorithm corresponds to a test.

In some systems, a job can be executed in two different modes, a *safe* mode and an *alternative* mode. The safe mode is always possible. The alternative mode may have a shorter processing time, but is not possible for every job. A test is necessary to determine whether the alternative mode is possible for a job and what the processing time in the alternative mode would be.

As a final application area, consider settings where a diagnosis can be carried out to determine the exact processing time of a job. For example, a fault diagnosis can determine the time needed for a repair job, or a medical diagnosis can determine the time needed for a consultation and treatment session with a patient. Assume that the resource that carries out the diagnosis is the same resource that executes the job (e.g., an engineer or a medical doctor), and that the resource must be allocated to a job for an uninterruptible period that is guaranteed to cover the actual time needed for the job. If the diagnosis takes unit time, we arrive at our problem of scheduling with testing.

Depending on the application under consideration, it may appear more natural to speak about 'compressing' or 'optimizing' rather than 'testing' a job, but for the sake of simplicity we use the term 'testing' throughout this paper.

In some applications, it may be appropriate to allow the time for testing a job to be different for different jobs (e.g., proportional to the upper limit of a job). We leave the consideration of such generalizations of the problem to future work.

Related work. Scheduling with testing can be viewed as a problem in the area of explorable (or queryable) uncertainty, where additional information about the input can be learned using a query operation (in our case, a test). The line of research on optimization with explorable uncertain data has been initiated by Kahan [8] in 1991. His work concerns selection problems with the goal of minimizing the number of queries that are necessary to find the optimal solution. Later, other problems studied in this uncertainty model include finding the k-th smallest value in a set of uncertainty intervals [8, 7, 5] (also with non-uniform query cost [5]), caching problems in distributed databases [12], computing a function value [9], and classical combinatorial optimization problems, such as shortest path [4], finding the median [5], the knapsack problem [6], and the MST problem [3, 11]. While most work aims for minimal query sets to guarantee exact optimal solutions, Olsten and Widom [12] initiate the study of trade-offs between the number of queries and the precision of the found solution. They are concerned with caching problems. Further work in this vein can be found in [9, 4, 5].

In all this previous work, the execution of queries is separate from the actual optimization problem being solved. In our case, the tests are executed by the same machine that runs the jobs. Hence, the tests are not considered separately, but they directly affect the objective value of the actual problem (by delaying the completion of other jobs while a job is being tested). Hence, instead of minimizing the number of tests needed until an optimal schedule can be computed (which would correspond to the standard approach in the work on explorable uncertainty discussed above), in our case the tests of jobs are part of the schedule, and we are interested in the sum of completion times as the single objective function.

Our adversarial model is inspired by (and draws motivation from) recent work on a stochastic model of scheduling with testing introduced in [10, 13]. They consider the problem of minimizing the weighted sum of completion times on one machine for jobs whose processing times and weights are random variables with a joint distribution, and are independent and identically distributed across jobs. In their model, testing a job does not make its processing time shorter, it only provides information for the scheduler (by revealing the exact weight and processing time for a job, whereas initially only the distribution is known). They present structural results about optimal policies and efficient optimal or near-optimal solutions based on dynamic programming.

Our contribution. A scheduling algorithm in the model of explorable uncertainty has to make two types of decisions: which jobs should be tested, and in what order should job executions and tests be scheduled. There is a subtle compromise to be found between

Table 1 New contributions for minimizing the sum of completion times. * holds asymptotically

competitive ratio	lower bound	upper bound	
deterministic algorithms	1.8546	2	THRESHOLD
randomized algorithms	1.6257	1.7453	RANDOM
det. alg. on uniform instances	1.8546	1.9338*	BEAT
det. alg. on extreme uniform instances	1.8546	1.8668	UTE
det. alg. on extreme uniform inst. with $\bar{p} \approx 1.9896$	1.8546	1.8552	UTE

investing time to test jobs and the benefit one can gain from these tests. We design scheduling algorithms that address this exploration-exploitation question in different ways and provide nearly tight bounds on the competitive ratio. In our analysis, we first show that worst-case instances have a particular structure that can be described by only a few parameters. This goes hand in hand with analyzing also the structure of both, an optimal and an algorithm's schedule. Then we express the total cost of both schedules as functions of these few parameters. It is noteworthy that, under the assumptions made, we typically characterize the *exact* worst-case ratio. Given the parameterized cost ratio, we analyze the worst-case parameter choice. This technical part involves second order analysis which can be performed with computer assistance. We use the algebraic solver Mathematica.

Our results are the following. For scheduling with testing on a single machine with the objective of minimizing the sum of completion times, we present a 2-competitive deterministic algorithm and prove that no deterministic algorithm can achieve competitive ratio less than 1.8546. We then present a 1.7453-competitive randomized algorithm, showing that randomization provably helps for this problem. We also give a lower bound of 1.626 on the best possible competitive ratio of any randomized algorithm. Both lower bounds hold even for instances with uniform upper limits where every processing time is either 0 or equal to the upper limit. We call such instances *extreme uniform instances*. For such instances we give a 1.8668-competitive algorithm. In the special case where the upper limit of all jobs is ≈ 1.9896, the value used in our deterministic lower bound construction, that algorithm is even 1.8552-competitive. For the case of uniform upper limits and arbitrary processing times, we give a deterministic 1.9338-competitive algorithm. An overview of these results is shown in Table 1. Finally, we also mention some results for the simpler problem of minimizing the makespan in scheduling with testing.

Omitted proofs can be found in a full version of the paper; see the preprint [2]. Computations performed with the support of Mathematica are provided as notebook- and pdf-files at a companion webpage.[1]

2 Preliminaries

Problem definition. The problem of scheduling with testing is defined as follows. We are given n jobs to be scheduled on a single machine. Each job j has an upper limit $\bar{p}_j$. It can either be executed untested (taking time $\bar{p}_j$), or be tested (taking time 1) and then executed at an arbitrary later time (taking time p_j, where $0 \leq p_j \leq \bar{p}_j$). Initially only $\bar{p}_j$ is known for each job, and p_j is only revealed after j is tested. The machine can either test or execute a job at any time. The completion time of job j is denoted by C_j. Unless noted otherwise, we consider the objective of minimizing the sum of completion times $\sum_j C_j$.

[1] `http://cslog.uni-bremen.de/nmegow/public/mathematica-SwT.zip`

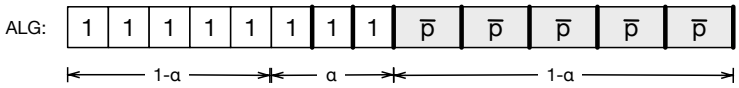

Figure 1 Typical schedule produced by an algorithm. Jobs are grey, tests are white. The completion time of a job is depicted by a thick bar. A job of length 0 completes immediately after testing.

Performance analysis. We compare the performance of an algorithm ALG to the optimal schedule using competitive analysis [1]. We denote by $\text{ALG}(I)$ the objective value (cost) of the schedule produced by ALG for an instance I, and by $\text{OPT}(I)$ the optimal cost. An algorithm ALG is *ρ-competitive* or *has competitive ratio at most ρ* if $\text{ALG}(I)/\text{OPT}(I) \leq \rho$ for all instances I of the problem. For randomized algorithms, $\text{ALG}(I)$ is replaced by $E[\text{ALG}(I)]$ in this definition.

When we analyze an algorithm or the optimal schedule, we will typically first argue that the schedule has a certain structure with different blocks of tests or job completions. Once we have established that structure, the cost of the schedule can be calculated by adding the cost for each block taken in isolation, plus the effect of the block on the completion times of later jobs. For example, assume that we have n jobs with upper limit $\bar{p}$, that αn of these jobs are *short*, with processing time 0, and $(1-\alpha)n$ jobs are *long*, with processing time $\bar{p}$. If an algorithm (in the worst case) first tests the $(1-\alpha)n$ long jobs, then tests the αn short jobs and executes them immediately, and finally executes the $(1-\alpha)n$ long jobs that were tested earlier (see also Figure 1), the total cost of the schedule can be calculated as

$$(1-\alpha)n^2 + \frac{\alpha n(\alpha n+1)}{2} + \alpha n(1-\alpha)n + \frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p}$$

where $(1-\alpha)n^2$ is the total delay that the $(1-\alpha)n$ tests of long jobs add to the completion times of all n jobs, $\frac{\alpha n(\alpha n+1)}{2}$ is the sum of completion times of a block with αn short jobs that are tested and executed, $\alpha n(1-\alpha)n$ is the total delay that the block of short jobs with total length αn adds to the completion times of the $(1-\alpha)n$ jobs that come after it, and $\frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p}$ is the sum of completion times for a block with $(1-\alpha)n$ job executions with processing time $\bar{p}$ per job.

Lower limits. A natural generalization of the problem would be to allow each job j to have, in addition to its upper limit $\bar{p}_j$, also a lower limit ℓ_j, such that the processing time after testing satisfies $\ell_j \leq p_j \leq \bar{p}_j$. We observe that the presence of lower limits has no effect on the optimal schedule, and can only help an algorithm. As we are interested in worst-case analysis, we assume in the remainder of the paper that every job has a lower limit of 0. Any algorithm that is ρ-competitive in this case is also ρ-competitive in the case with arbitrary lower limits (the algorithm can simply ignore the lower limits).

Jobs with small $\bar{p}_j$. We will consider several algorithms and prove competitiveness for them. We observe that any ρ-competitive algorithm may process jobs with $\bar{p}_j < \rho$ without testing in order of increasing $\bar{p}_j$ at the beginning of its schedule.

▶ **Lemma 1.** *Without loss of generality any algorithm* ALG *(deterministic or randomized) claiming competitive ratio ρ starts by scheduling untested all jobs j with $\bar{p}_j < \rho$ in increasing order of $\bar{p}_j$. Also worst-case instances for* ALG *consist solely of jobs j with $\bar{p}_j \geq \rho$.*

Increasing or decreasing Alg and Opt. Throughout the paper we sometimes consider worst-case instances consisting of only a few different job types. The following proposition allows us to do so in some cases.

▶ **Proposition 2.** *Fix some algorithm* Alg *and consider a family of instances described by some parameter $x \in [\ell, u]$, which could represent p_j or $\bar{p}_j$ for some job j or for some set of jobs. Suppose that both* Opt *and* Alg *are linear in x for the range $[\ell, u]$. Then the ratio* Alg/Opt *does not decrease for at least one of the two choices $x = \ell$ or $x = u$. Moreover, if* Opt *and* Alg *are increasing in x with the same slope, then this holds for $x = \ell$.*

We can make successive use of this proposition in order to show useful properties on worst-case instances. For this purpose we say that a schedule is *insensitive* to changes of the processing times, if the order of tests and job executions does not change, even though some completion times could be shifted.

▶ **Lemma 3.** *Suppose that there is an interval $[\ell', u']$ such that* Opt *schedules all jobs j with $p_j \in [\ell', u']$ either all tested or all untested, independently of the actual processing time in $[\ell', u']$. Suppose that this holds also for* Alg*. Moreover suppose that both* Opt *and* Alg *are insensitive to changes of the processing times in $[\ell', u']$ which maintain the ordering of processing times. Then there is a worst-case instance for* Alg *where every job j with $p_j \in [\ell', u']$ satisfies $p_j \in \{\ell', u'\}$.*

Proof sketch. The proof consists of fixing some set of jobs S of the same processing time and identifying an interval $[\ell, u]$ around it with $[\ell, u] \subseteq [\ell', u']$. Then we can show that both costs Opt and Alg are linear in x when changing the processing times of the jobs in S to any value $x \in [\ell, u]$. Proposition 2 implies that choosing $x \in \{\ell, u\}$ does not decrease the competitive ratio. This argument can be repeated until the number of distinct processing times in the open interval (ℓ', u') decreases to zero. ◀

3 Deterministic Algorithms

3.1 Algorithm Threshold

We show a competitive ratio of 2 for a natural algorithm that uses a threshold to decide whether to test a job or execute it untested.

▶ **Algorithm** (Threshold). *First jobs with $\bar{p}_j < 2$ are scheduled in order of non-decreasing upper limits without testing. Then all remaining jobs are tested. If the revealed processing time of job j is $p_j \leq 2$ (short jobs), then the job is executed immediately after its test. After all pending jobs (long jobs) have been tested, they are scheduled in order of increasing processing time p_j.*

By Lemma 1 we may restrict our analysis w.l.o.g. to instances with $\bar{p}_j \geq 2$. Note, that on such instances Threshold tests all jobs. From a simple interchange argument it follows that the structure of the algorithm's solution in a worst-case instance is as follows.

- The algorithm tests all jobs that have $p_j > 2$, and defers them.
- The algorithm tests short jobs ($p_j \leq 2$) and executes each of them right away. The jobs are tested in order of non-increasing processing time.
- The algorithm executes all deferred long jobs in order of non-decreasing processing times.

An optimal solution will not test jobs with $p_j + 1 \geq \bar{p}_j$. It sorts jobs in non-decreasing order of values $\min\{1 + p_j, \bar{p}_j\}$.

First, we analyze and simplify worst-case instances.

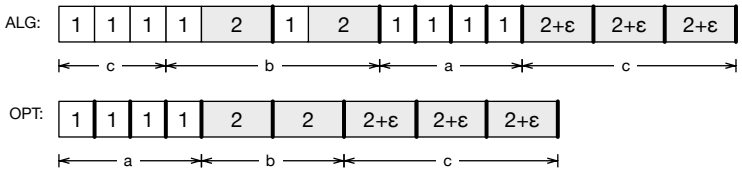

Figure 2 Worst-case instance for THRESHOLD.

▶ **Lemma 4.** *There is a worst-case instance for* THRESHOLD *in which all short jobs with* $p_j \leq 2$ *have processing time either* 0 *or* 2.

▶ **Lemma 5.** *There is a worst-case instance in which long jobs with* $p_j > 2$ *satisfy* $p_j = \bar{p}_j = 2 + \epsilon$ *for infinitesimally small* $\epsilon > 0$.

Now we are ready to prove the main result.

▶ **Theorem 6.** *Algorithm* THRESHOLD *has competitive ratio at most* 2.

Proof. We consider worst-case instances given by Lemmas 4 and 5. Let a be the number of short jobs with $p_j = 0$, let b be the number of short jobs with $\bar{p}_j = p_j = 2$, and let c be the number of long jobs with $\bar{p}_j = 2 + \epsilon$, see Figure 2.

THRESHOLD's solution for a worst-case instance first tests all long jobs, then tests and executes the short jobs in decreasing order of processing times, and completes with the executions of long jobs. The total objective value ALG is

$$\text{ALG} = (a+b+c)c + b(b+1)/2 \cdot 3 + 3b(a+c) + a(a+1)/2 + a \cdot c + c(c+1)/2 \cdot (2+\epsilon).$$

An optimum solution tests and schedules first all 0-length jobs and then executes the remaining jobs without tests. The objective value is

$$OPT = a(a+1)/2 + a(b+c) + b(b+1)/2 \cdot 2 + 2bc + c(c+1)/2 \cdot (2+\epsilon).$$

Simple transformation shows that $\text{ALG} \leq 2 \cdot OPT$ is equivalent to

$$2ab + 2c^2 \leq a^2 + b^2 + a + b + c(c+1)(2+\epsilon) \;\Leftrightarrow\; 0 \leq (a-b)^2 + a + b + c^2\epsilon + c(2+\epsilon),$$

which is obviously satisfied and the theorem follows. ◀

3.2 Deterministic lower bound

In this section we give a lower bound on the competitive ratio of any deterministic algorithm. The instances constructed by the adversary have a very special form: All jobs have the same upper limit $\bar{p}$, and the processing time of every job is either 0 or $\bar{p}$.

Consider instances of n jobs with uniform upper limit $\bar{p} > 1$, and consider any deterministic algorithm. We say that the algorithm *touches* a job when it either tests the job or executes it untested. We re-index jobs in the order in which they are first touched by the algorithm, i.e., job 1 is the first job touched by the algorithm and job n is the last. The adversary fixes a fraction $\delta \in [0,1]$ and sets the processing time of job j, $1 \leq j \leq n$, to:

$$p_j = \begin{cases} 0 & \text{, if } j \text{ is executed by the algorithm untested, or } j > \delta n \\ \bar{p} & \text{, if } j \text{ is tested by the algorithm and } j \leq \delta n \end{cases}$$

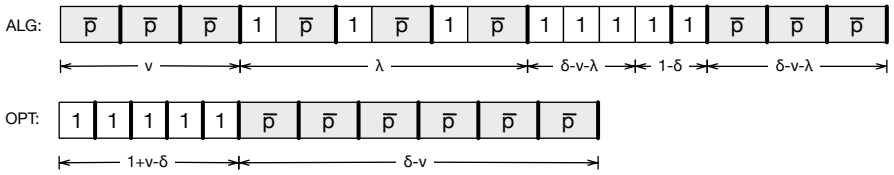

Figure 3 Lower bound construction.

A job j is called *short* if $p_j = 0$ and *long* if $p_j = \bar{p}$. Let j_0 be the smallest integer that is greater than δn. Job j_0 is the first of the last $(1-\delta)n$ jobs, which are short no matter whether the algorithm tests them or not.

We assume the algorithm knows $\bar{p}$ and δ, which can only improve the performance of the best-possible deterministic algorithm. Note that with δ and $\bar{p}$ known to the algorithm, it has full information about the actions of the adversary. Nevertheless, it is still non-trivial for an algorithm to decide for each of the first δn jobs whether to test it (which makes the job a long job, and hence the algorithm spends time $\bar{p}+1$ on it while the optimum executes it untested and spends only time $\bar{p}$) or to execute it untested (which makes it a short job, and hence the algorithm spends time $\bar{p}$ on it while the optimum spends only time 1).

Let us first determine the structure of the schedule produced by an algorithm that achieves the best possible competitive ratio for instances created by this adversary, as displayed in Figure 3.

▶ **Lemma 7.** *The schedule of a deterministic algorithm with best possible competitive ratio has the following form, where $\lambda, \nu \geq 0$ and $\nu + \lambda \leq \delta$: The algorithm first executes νn jobs untested, then tests and executes λn long jobs, then tests $(\delta - \nu - \lambda)n$ long jobs and delays their execution, then tests and executes the remaining $(1-\delta)n$ short jobs, and finally executes the $(\delta - \nu - \lambda)n$ delayed long jobs that were tested earlier.*

The cost of the algorithm in dependence on ν, λ, δ and $\bar{p}$ can now be expressed as:

$$\begin{aligned}\textsc{Alg}(\nu,\lambda,\delta,\bar{p}) &= n^2\big(\frac{\nu^2}{2}\bar{p} + \nu\bar{p}(1-\nu) + \frac{\lambda^2}{2}(1+\bar{p}) + \lambda(1+\bar{p})(1-\nu-\lambda) \\ &\quad + (\delta-\nu-\lambda)(1-\nu-\lambda) + \frac{(1-\delta)^2}{2} + (1-\delta)(\delta-\nu-\lambda) + \frac{(\delta-\nu-\lambda)^2}{2}\bar{p}\big) + O(n) \\ &= \frac{n^2}{2}(1 + 2\delta(1-\nu\bar{p}) + \delta^2(\bar{p}-1) + 2\nu(\nu+\bar{p}-2) + \lambda^2 + 2\lambda(\nu+\bar{p}-1-\delta\bar{p})) + O(n)\end{aligned}$$

The optimal schedule first tests and executes the $(\nu+1-\delta)n$ short jobs and then executes the $(\delta-\nu)n$ long jobs untested. Hence, the optimal cost, which depends only on ν, δ and $\bar{p}$, is:

$$\begin{aligned}\textsc{Opt}(\nu,\delta,\bar{p}) &= n^2\left(\frac{(\nu+1-\delta)^2}{2} + (\nu+1-\delta)(\delta-\nu) + \frac{(\delta-\nu)^2}{2}\bar{p}\right) + O(n) \\ &= \frac{n^2}{2}(1 + (\delta-\nu)^2(\bar{p}-1)) + O(n)\end{aligned}$$

Let $\textsc{Alg}'(\nu,\lambda,\delta,\bar{p}) = \lim_{n\to\infty}\frac{1}{n^2}\textsc{Alg}(\nu,\lambda,\delta,\bar{p})$ and $\textsc{Opt}'(\nu,\delta,\bar{p}) = \lim_{n\to\infty}\frac{1}{n^2}\textsc{Opt}(\nu,\delta,\bar{p})$. As the adversary can choose δ and $\bar{p}$, while the algorithm can choose ν and λ, the value

$$R = \max_{\delta,\bar{p}}\min_{\nu,\lambda}\frac{\textsc{Alg}'(\nu,\lambda,\delta,\bar{p})}{\textsc{Opt}'(\nu,\delta,\bar{p})}$$

gives a lower bound on the competitive ratio of any deterministic algorithm in the limit for $n \to \infty$. By making n sufficiently large, the adversary can create instances with finite n that give a lower bound that is arbitrarily close to R.

The exact optimization of δ and $\bar{p}$ is rather technical as it involves the optimization of rational functions of several variables. The choices $\delta = 0.6306655$ and $\bar{p} = 1.9896202$ give a lower bound of 1.854628. (The fully optimized value of R is less than $1.1 \cdot 10^{-7}$ larger.)

▶ **Theorem 8.** *No deterministic algorithm can achieve a competitive ratio below* 1.854628. *This holds even for instances with uniform upper limit where each processing time is either* 0 *or equal to the upper limit.*

4 Randomized Algorithms

4.1 Algorithm Random

▶ **Algorithm** (RANDOM). *The algorithm has parameters* $1 \leq T \leq E$ *and works in 3 phases. First it executes all jobs with* $\bar{p}_j < T$ *without testing in order of increasing* $\bar{p}_j$. *Then it tests all jobs with* $\bar{p}_j \geq T$ *in uniform random order. Each tested job* j *is executed immediately after its test if* $p_j \leq E$ *and is deferred otherwise. Finally all deferred jobs are executed in order of increasing processing-time.*

We analyze the competitive ratio of RANDOM, and optimize the parameters T, E such that the resulting competitive ratio is T.

▶ **Theorem 9.** *The competitive ratio of the algorithm* RANDOM *is at most* 1.7453.

Proof sketch. By Lemma 1 we restrict to instances with $\bar{p}_j \geq T$ for all jobs. Then, the schedule produced by RANDOM can be divided into two parts, see Figure 4. Part (1) contains all tests, of which those that yield processing time p_j at most E are immediately followed by the job's execution. Part (2) contains all jobs that are tested and have processing time larger than E. These jobs are ordered by increasing processing time. Jobs in the first part are completed in an arbitrary order.

Furthermore, we can assume $\bar{p}_j = \max\{p_j, T\}$ for all jobs. Reducing $\bar{p}_j$ to this value does not change the cost or behavior of RANDOM, but may decrease the cost of OPT. We make extensive use of Proposition 2 to show that the worst-case instance consists of only 4 types of jobs. As a result the adversary chooses fractions $\alpha, \beta, \gamma \geq 0$ (of total sum at most 1) describing the following instance.

- A $1 - \alpha - \beta - \gamma$ fraction of the jobs have $\bar{p}_j = T$ and $p_j = 0$.
- An α fraction of the jobs have $\bar{p}_j = T$ and $p_j = T$.
- A β fraction of the jobs have $\bar{p}_j = E$ and $p_j = E$.
- A γ fraction of the jobs have $\bar{p}_j = E + \epsilon$ and $p_j = E + \epsilon$ for some arbitrarily small $\epsilon > 0$.

In this setting the cost of RANDOM as well as the optimal cost can be expressed as a function in $\alpha, \beta, \gamma, T, E, n$ each consisting of two parts: one multiple of n^2 and the other multiple of n. We start to analyze the ratio only for the first part, and verify that the ratio also holds for the second part.

For the analysis we consider the expression $g := T \cdot \text{OPT} - \text{ALG}$ which is non-negative iff the ratio is at most T. Hence the adversary chooses fractions α, β, γ which minimize g. Our general approach is therefore to identify local minima for g inside the polytope $\{(\alpha, \beta, \gamma) | \alpha, \beta, \gamma \geq 0, \alpha + \beta + \gamma \leq 1\}$. This is done by standard second order analysis with the help of Mathematica, distinguishing the cases when such a minimum lies in the inner region

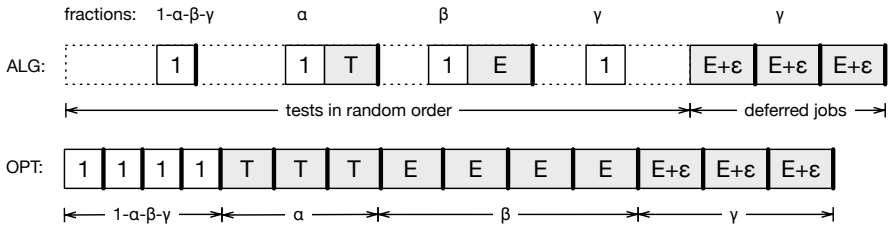

Figure 4 Worst-case analysis of the algorithm RANDOM.

of the polytope, or on one of the 4 triangular shaped facets or on one of the 6 one-dimensional edges. Each of these minima generates inequalities for T, E, generated by $g \geq 0$, which the algorithm needs to respect if the ratio is to be at most T in all cases. Our proof identifies the critical inequalities, and optimizes T, E for them, obtaining algebraic values, which are approximately $T \approx 1.7453$ and $E \approx 2.8609$. ◀

4.2 Lower bound for randomized algorithms

In this section we give a lower bound on the best possible competitive ratio of any randomized algorithm against an oblivious adversary. We do so by specifying a probability distribution over inputs and proving a lower bound on $E[\text{ALG}]/E[\text{OPT}]$ that holds for all deterministic algorithms ALG. By Yao's principle [14, 1] this gives the desired lower bound.

The probability distribution over inputs with n jobs has a constant parameter $0 < q < 1$ and is defined as follows: Each job j has upper limit $\bar{p}_j = 1/q > 1$, and its processing time p_j is set to 0 with probability q and to $1/q$ with probability $1 - q$. We show the expected optimal cost is

$$E[\text{OPT}] = \frac{n^2}{2}\left(\frac{1}{q} + 3q - 2 - q^2\right) + O(n).$$

We determine the algorithm cost by induction. With a case distinction on how the algorithm handles the first job, if it tests or executes it, we show the expected algorithm cost is at least $E[\text{ALG}(n)] \geq \frac{n^2}{2q}$. For $q = 1 - 1/\sqrt{3} \approx 0.4227$, this yields a lower bound of 1.6257 by Yao's principle.

▶ **Theorem 10.** *No randomized algorithm can achieve a competitive ratio less than* 1.6257.

5 Deterministic Algorithms for Uniform Upper Limits

5.1 An improved algorithm for uniform upper limits

In this section we present an algorithm for instances with uniform upper limit $\bar{p}$ that achieves a ratio strictly less than 2. We present a new algorithm BEAT that performs well on instances with upper limit roughly 2, but its performance becomes worse for larger upper limits. Thus, in this case we employ the algorithm THRESHOLD presented in Section 3.1.

To simplify the analysis, we consider the limit of $\text{ALG}(I)/\text{OPT}(I)$ when the number n of jobs approaches infinity. We say that an algorithm ALG is *asymptotically ρ_∞-competitive* or *has asymptotic competitive ratio at most ρ_∞* if $\lim_{n\to\infty} \sup_I \text{ALG}(I)/\text{OPT}(I) \leq \rho_\infty$.

▶ **Algorithm** (BEAT). *The algorithm* BEAT *balances the time testing jobs and the time executing jobs while there are untested jobs. A job is called* short *if its running time is at*

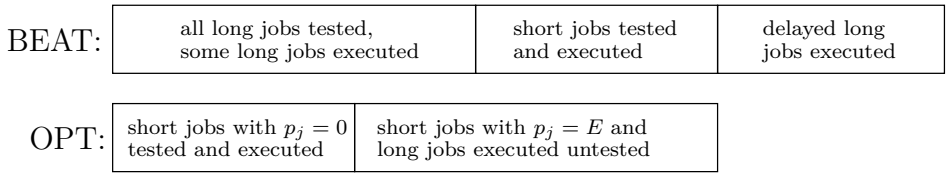

Figure 5 Structure of schedules produced by BEAT and OPT.

most $E = \max\{1, \bar{p} - 1\}$*, and* long *otherwise. Let TotalTest denote the time we spend testing long jobs and let TotalExec be the time long jobs are executed. We iterate testing an arbitrary job and then execute the job with smallest processing time either, if it is a short job, or if* $\text{TotalExec} + p_k$ *is at most TotalTest. Once all jobs have been tested, we execute the remaining jobs in order of non-decreasing processing time.*

We make a structural observation about the algorithm schedule for a worst-case instance.

▶ **Lemma 11.** *The adversary gives jobs with* $p_j \in \{0, E, \bar{p}\}$ *and at most one job with* $p_j \in (E, \bar{p})$ *in order of decreasing* p_j*.*

Consequently, the schedule produced by BEAT and the optimal schedule display a clear structure, which we depict in Figure 5.

We prove that the asymptotic competitive ratio of BEAT for $\bar{p} < 3$ is at most

$$\rho_{\infty}^{BEAT} = \frac{1 + 2(-2 + \bar{p})\bar{p} + \sqrt{(1 - 2\bar{p})^2(-3 + 4\bar{p})}}{2(-1 + \bar{p})\bar{p}},$$

which is a function decreasing in $\bar{p}$.

For small upper limit we can execute each job without test. Then there is a worst-case instance where all jobs have processing time $p_j = 0$. The optimal schedule tests each job only if the upper limit $\bar{p}$ is larger than 1 and executes it immediately. For $\bar{p} < 1$ this means the competitive ratio is 1 and otherwise it is $\bar{p}$, which monotonically increases. Thus, we choose a threshold $T_1 \approx 1.9338$ for $\bar{p}$, deciding if we run jobs untested or apply BEAT. T_1 is the fixpoint of the function ρ_{∞}^{BEAT}.

For upper limit $\bar{p} > 3$, the performance behavior of BEAT changes and the asymptotic competitive ratio increases. Thus, we employ the algorithm THRESHOLD for large upper limits. Recall that for $\bar{p} > 2$ THRESHOLD tests all jobs, executes those with $p_j \leq 2$ immediately and defers the other jobs. By Lemma 4, there is a worst-case instance with short jobs that have processing time either 0 or 2. Moreover, we argue that in a worst case long jobs have processing time $\bar{p}_j = \bar{p}$ and that no long job is tested in an optimal solution. This allows us to prove

$$\rho_{\infty}^{THRESH} = \begin{cases} \frac{-3+\bar{p}+\sqrt{-15+\bar{p}(18+\bar{p})}}{2(\bar{p}-1)} & \text{if } \bar{p} \in (2,3) \\ \sqrt{3} \approx 1.73 & \text{if } \bar{p} \geq 3. \end{cases}$$

The function for small $\bar{p}$ is a monotone function decreasing from 2 to $\sqrt{3}$ in the limits for $\bar{p} \in (2, 3)$. We choose a threshold for $\bar{p}$, where we change from applying BEAT to employing THRESHOLD at $T_2 \approx 2.2948$, the crossing point of the two functions describing the competitive ratio of BEAT and THRESHOLD in $(2, 3)$.

▶ **Algorithm.** *Execute all jobs without test, if the upper limit* $\bar{p}$ *is less than* $T_1 \approx 1.9338$*. Otherwise, if the upper limit* $\bar{p}$ *is greater than* $T_2 \approx 2.2948$*, execute the algorithm* THRESHOLD*. For all upper limits between* T_1 *and* T_2*, execute the algorithm* BEAT*.*

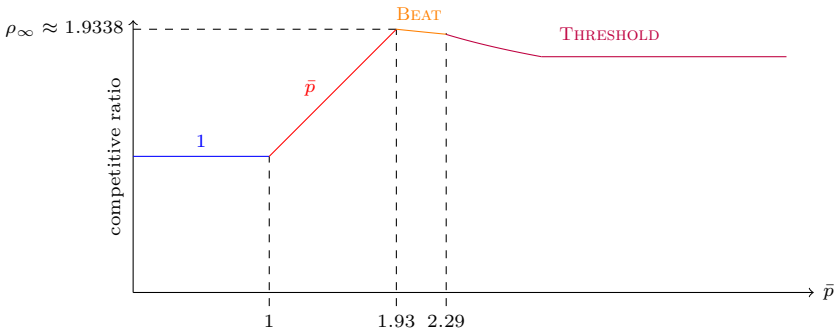

Figure 6 Competitive ratio depending on $\bar{p}$.

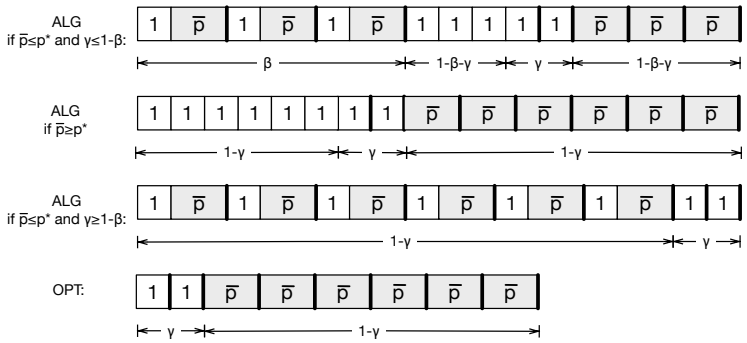

Figure 7 The schedule produced by UTE and the optimal schedule.

The function describing the asymptotic competitive ratio depending on $\bar{p}$ is displayed in Figure 6. Its maximum is attained at T_1, which is a fixpoint. Thus we have

▶ **Theorem 12.** *The asymptotic competitive ratio of our algorithm is* $\rho_\infty = T_1 \approx 1.9338$, *which is the only real root of* $2\bar{p}^3 - 4\bar{p}^2 + 4\bar{p} - 1 - \sqrt{(1-2\bar{p})^2(4\bar{p}-3)}$.

5.2 Nearly tight deterministic algorithm for extreme uniform instances

We present a deterministic algorithm for the class of *extreme uniform* instances, that is almost tight for the instance that yields the deterministic lower bound. An *extreme uniform* instance consists of jobs with uniform upper limit $\bar{p}$ and processing times in $\{0, \bar{p}\}$. Our algorithm UTE attains asymptotic competitive ratio $\rho_\infty \approx 1.8668$ for this class of instances.

▶ **Algorithm** (UTE). *If the upper limit* $\bar{p}$ *is at most* ρ, *then all jobs are executed without test. Otherwise, all jobs are tested. The first* $\max\{0, \beta\}$ *fraction of the jobs are executed immediately after their test. The remaining fraction of the jobs are executed immediately after their test if they have processing time* 0 *and are delayed otherwise, see Figure 7. The parameter* β *is defined as*

$$\beta = \frac{1 - \bar{p} + \bar{p}^2 - \rho + 2\bar{p}\rho - \bar{p}^2\rho}{1 - \bar{p} + \bar{p}^2 - \rho + \bar{p}\rho}. \tag{1}$$

▶ **Theorem 13.** *The competitive ratio of* UTE *is at most* $\rho = \frac{1+\sqrt{3+2\sqrt{5}}}{2} \approx 1.8668$.

Proof sketch. An instance is defined by the job number n, an upper limit $\bar{p}$ and a fraction γ such that the first $1-\gamma$ fraction of the jobs tested by UTE have processing time $\bar{p}$, while the jobs in the remaining γ fraction have processing time 0. The algorithm chooses β so as to have the smallest ratio ρ.

First we observe that there is a value p^* such that $\beta \geq 0$ only when $\bar{p} \leq p^*$. Then we analyze the competitive ratio of UTE, distinguishing the cases $\bar{p} \leq \rho$ (covered by Lemma 1), $\bar{p} \geq p^*$ and $\bar{p} \leq p^*$. The last case is furthermore subdivided into cases $\gamma \geq 1-\beta$ and $\gamma \leq 1-\beta$. For each of these cases we use first and second order analysis to determine the worst values $\bar{p}$ and γ for the ratio, and the best response β the algorithm can choose, optimizing β and ρ on the way, and obtaining the claimed values. ◀

▶ Remark. The deterministic lower bound 1.8546 in Theorem 8 uses the upper limit $\bar{p} \approx 1.9896$. Plugging this choice of $\bar{p}$ into a precise form of the competitive ratio which we obtained in the proof of the theorem, we can show that UTE has asymptotic competitive ratio $\rho_\infty \approx 1.8552$ on this instance. This is almost tight.

6 Optimal Testing for Minimizing the Makespan

We consider scheduling with testing with the objective of minimizing the makespan, i.e., the completion time of the last job. For this problem we give the best possible competitive ratio for deterministic and randomized algorithms. The key insight is that for any algorithm that treats each job independent of its position in the schedule, there is a worst-case instance containing only a single job. The reason is that the execution of a job (possibly including testing) has a linear contribution to the makespan.

▶ **Theorem 14.** *Let $\varphi \approx 1.618$ be the golden ratio. Testing each job j if and only if $\bar{p}_j > \varphi$ is an algorithm with competitive ratio φ. This is best possible for deterministic algorithms.*

▶ **Theorem 15.** *The randomized algorithm that tests each job with $\bar{p}_j > 1$ with probability $1 - 1/(\bar{p}_j^2 - \bar{p}_j + 1)$ has competitive ratio $4/3$. No randomized algorithm can achieve a better competitive ratio against an oblivious adversary.*

7 Conclusion

In this paper we have introduced an adversarial model of scheduling with testing where a test can shorten a job but the time for the test also prolongs the schedule, thus making it difficult for an algorithm to find the right balance between tests and executions. We have presented upper and lower bounds on the competitive ratio of deterministic and randomized algorithms for a single-machine scheduling problem with the objective of minimizing the sum of completion times or the makespan. An immediate open question is whether it is possible to achieve competitive ratio below 2 for minimizing the sum of completion times with a deterministic algorithm for arbitrary instances. Further interesting directions for future work include the consideration of job-dependent test times or other scheduling problems such as parallel machine scheduling or flow shop problems. More generally, the study of problems with explorable uncertainty in settings where the costs for querying uncertain data directly contribute to the objective value is a promising direction for future work.

References

1 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

2 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. Scheduling with explorable uncertainty. *CoRR*, abs/1709.02592, 2017. arXiv:1709.02592.

3 Thomas Erlebach, Michael Hoffmann, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, pages 277–288, 2008.

4 Tomás Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007.

5 Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003.

6 Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Computers & OR*, 55:12–22, 2015.

7 Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. The update complexity of selection and related problems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 325–338. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

8 Simon Kahan. A model for data in motion. In *23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*, pages 267–277, 1991.

9 Sanjeev Khanna and Wang-Chiew Tan. On computing functions with uncertainty. In *20th Symposium on Principles of Da18th Annual Symposium on Database Systems (PODS 2001)*, pages 171–182, 2001.

10 Retsev Levi. Practice driven scheduling models. Talk at Dagstuhl Seminar 16081: Scheduling, 2016.

11 Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. In *Algorithms – Proceedings of the 23rd Annual European Symposium (ESA 2015)*, LNCS 9294, pages 878–890. Springer, 2015.

12 Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 144–155, 2000.

13 Yaron Shaposhnik. *Exploration vs. Exploitation: Reducing Uncertainty in Operational Problems*. PhD thesis, Sloan School of Management, MIT, 2016.

14 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 222–227. IEEE, 1977.